LA-UR-19-29712
Approved for public release; distribution is unlimited.

Title: Asynchronous Navier-Stokes Solver on 3D Unstructured Grids for the Exascale Era

Author(s): Bakosi, Jozsef; Bird, Robert Francis; Junghans, Christoph; Pandare, Aditya Kiran; Pavel, Robert Stephen; Waltz, Jacob I.; Li, Weizhao; Luo, Hong; Bohm, Eric; Kale, Laxmikant; Mikida, Eric; Ramos, Evan; Barnett, Joshua; Collins, Gary; Pakki, Aditya

Intended for: Report
Web

Issued: 2019-09-26

Asynchronous Navier-Stokes Solver on 3D Unstructured Grids for the Exascale Era

Final Report for LDRD-20170127-ER, LA-UR-XX-XXXXX
September 24, 2019

J. Bakosi, R. Bird, C. Junghans, A. Pandare, R. Pavel, J. Waltz, Los Alamos National Laboratory
W. Li, H. Luo, North Carolina State Universtiy
E. Bohm, L. Kale, E. Mikida, E. Ramos, Charmworks, Inc.
J. Barnett, Georgia Institute of Technology, G. Collins, University of Tennessee, A. Pakki, University of Utah

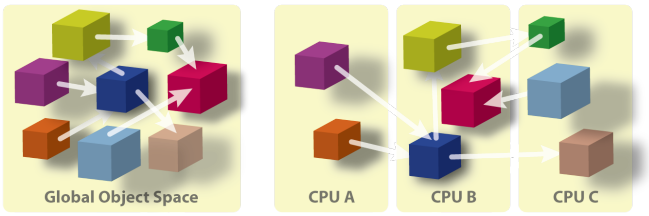
1 Summary

This project has developed multiple fluid dynamics solvers for complex 3D flows using fully asynchronous distributed-memory task-parallel algorithms on top of the Charm++ [1] runtime system. The algorithms solve the Euler or Navier-Stokes equations of compressible flows using unstructured tetrahedron meshes with optional solution-adaptive mesh-, and polynomial-order refinement. We have demonstrated excellent strong scaling up 50K compute cores and the benefits of Charm++'s automatic load balancing.

2 Accomplishments at a glance

- Implemented the first unstructured-mesh partial differential equations (PDE) solver on the Charm++ runtime system with automatic load balancing.
- Demonstrated, for the first time, that excellent parallel performance can be achieved using Charm++ of a PDE solver on unstructured grids, useful for complex 3D engineering problem geometries.
- Implemented both node-centered and cell-centered finite element algorithms for the simulation of compressible high-speed flows. All algorithms are in 3D, fully asynchronous, task-based, distributed-memory-parallel, and exhibit excellent strong scaling up to 50K CPU cores, the most tested.
- Developed and implemented a new adaptive DG algorithm that automatically adjusts the order of the approximation polynomial based on local error estimators and exercised it for single-material verification cases on 3D unstructured meshes with Charm++'s automatic load balancing capabilities.
- Developed, implemented, and verified a new DG method for compressible multi-material flows.
- Implemented adaptive mesh refinement in 3D using an, asynchronous distributed-memory-, task-parallel algorithm.
- Developed the code in a production fashion, with extensive unit-, and regression test suites and high-degree of code reuse using the C++17 standard. Also exercise mandatory code reviews, test code coverage analysis, using LANL-internal and public-facing continuous integration servers.
- Implemented various code capabilities that enable large-scale fluid dynamics, e.g., file/rank N-to-M parallel I/O, checkpoint/restart, and compile-time-configurable zero-runtime-overhead memory layout for large-data arrays to enable enable performance-portability across different architectures.
- Released the code as open source, see https://quinoacomputing.org.

3 A brief overview of Charm++



Interacting objects, (left) programmer's view, (right) the runtime system's view.

We have designed a fully asynchronous code on top of the Charm++ runtime system [1]. Although Charm++ is mature (20+ years) and used by several large production codes, including the Gordon Bell award winner molecular dynamics code, NAMD, with thousands of users, it is not currently explored for unstructured-mesh applications or within LANL or DOE. Charm++ is founded on the migratable-objects programming model and supported by an adaptive runtime system. In Charm++ data and work-units interact via asynchronous function calls which may reside on the same or on a networked compute node, and may migrate from one to another during computation. Object migration is based on real-time load measurement and is transparent to the application. The runtime system dynamically and automatically adapts the computational load monitoring load-imbalance and migrates data and work-units if it notices that a compute node failed or is about to fail, enabling fault tolerance. Since the underlying programming paradigm of Charm++ is higher level than MPI+X, in Charm++ distributed-memory sends and receives may operate on (safe) custom data structures (instead of low level byte-streams), e.g., containers built on C++ standard library vectors, trees, or hash maps, or arbitrary user-defined data types. We have demonstrated that communication using such high level data structures yields an unmeasurable performance hit when compared to useful computation (see figures from *Projections* below). The Charm++ paradigm is the single abstraction the code uses, and while still allow the reuse of existing MPI-only libraries, e.g., Zoltan, Hypre, ExodusII, HDF5, etc., it also allows the combination of fully asynchronous data and task parallelism, enabling arbitrary overlap of communication, computation, and I/O, enabling high performance.

We did not have to write any load balancing code. We simply write an algorithm and the runtime system automatically detects CPU load and homogenizes the computational load by migrating data.

4 Production-style code infrastructure

While not strictly required for the research and development set out in the proposal for this project, we believe that writing software in *production*-style, as opposed to *research*-style, is a good investment regarding the future utility of the code. Accordingly, we have invested a significant fraction of our time in software quality, modern development practices, testing, and generating easy-to-use documentation. We have set up a software development infrastructure with the following features and configuration:

- 102K source lines of *well-commented* code (every 3rd line is a comment)
- Building on 28 high-quality third-party libraries
- Unit-, and regression tests, routinely excercise 3 compilers (86% test code coverage)
- Open source: `https://github.com/quinoacomputing/quinoa`
- Latest releases as LANL copyright (re-)assertions at `https://github.com/quinoacomputing/quinoa/releases`
- Core Infrastructure Initiative Best Practices self-assessment at `https://bestpractices.coreinfrastructure.org/en/projects/2120`
- Detailed roadmap and contributing guide for open source collaboration
- Mandatory code review, github work-flow (crucial for efficient external collaboration)
- Continuous integration (build & test matrix) with Azure & TeamCity
- Continuous quantified test code coverage with Gcov & CodeCov.io
- Continuous quantified *documentation* coverage with CodeCov.io
- Continuous static analysis with CppCheck & SonarQube
- Continuous deployment (of binary releases) to DockerHub

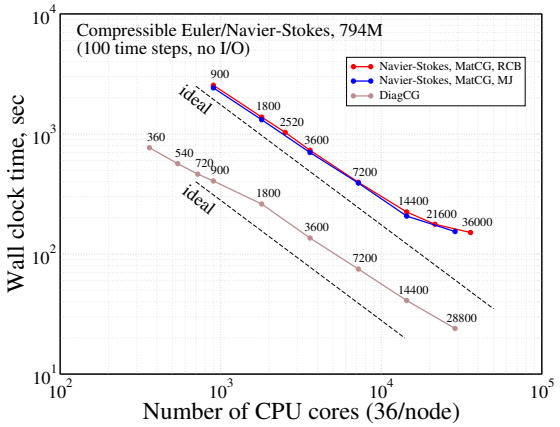
5 Hydrodynamics algorithms

This section gives a high-level overview of the hydrodynamics algorithms we have implemented during the course of this project to solve the Euler and Navier-Stokes equations. They are all finite element methods for 3D tetrahedra meshes.

5.1 Node-centered continuous Galerkin finite element method

In the interest of space the methods are not derived here, instead the reader is referred to [2], only some implementation details are given.

This method stores the solution unknowns at mesh points, as opposed to cell centers, and employs linear finite element shape functions for spatial approximation of the numerical solution across discrete elements. The spatial discretization results in a $N \times N$ linear system of algebraic equations, where N is the number of mesh nodes in the computational mesh. The system is integrated in time using an explicit Lax-Wendroff-type method, adopted to unstructured meshes. We verified that this method is second-order accurate for continuous solutions [3, 4]. For large gradients, shocks and other discontinuities, we also implemented a flux-corrected transport algorithm.



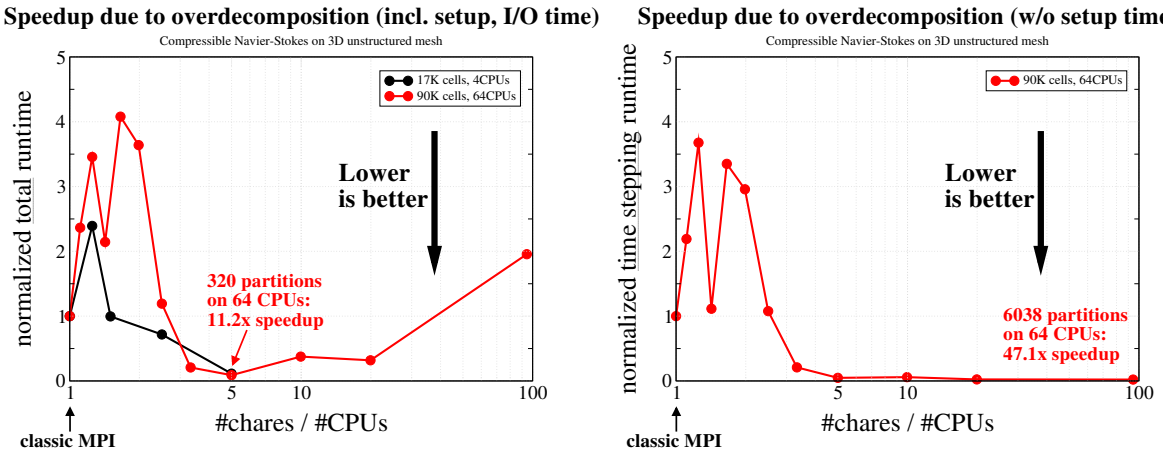
Strong scaling of the consistent-mass-matrix (MatCG) and lumped-mass-matrix (DiagCG) node-centered finite element solvers on *grizzly*. MJ and RCB denote two different types of initial mesh decomposition algorithms from the Zoltan2 library: MJ: multi-jagged, RCB: recursive coordinate bisection.

In parallel, the matrix is stored in a distributed fashion. Initially we stored each non-zero entry of the matrix in a compressed-row storage format, and solved it using a conjugate gradient algorithm from the hypre library available from LLNL. We wrote the matrix assembly in a fully asynchronous distributed-memory-parallel fashion relying on Charm++'s task-parallelism via their structured direct acyclic graph (SDAG) constructs. For more information on the asynchronous parallel matrix assembly, see [5]. To make this solver work, we successfully connected three MPI-only third-party libraries to our otherwise native Charm++ code: Zoltan2 (used for initial parallel mesh partitioning), hypre (used for solving distributed linear systems), and ExodusII (or SEACAS, used for parallel input of the computational mesh and output of the solution field data for visualization). Charm++/MPI interoperation allows reusing existing MPI-only libraries, although without Charm++'s automatic load balancing for the duration of the MPI library calls. The scalability of this matrix-based solver was reasonable (close to ideal up to about 20K CPU cores, but beyond that the frequent calls to parallel dot-products by the conjugate algorithm within hypre started to dominate and scalability was limited, see figure above. We then changed the solver to use a lumped mass matrix, which is not only much simpler to assemble and faster to solve, but also does not require a linear solver, so we eliminated hypre. This significantly improved scalability, see the figure above.

We have also experimented with *overdecomposition*, where the computational mesh is decomposed into more than the number of CPUs assigned. We have gained much insight from this, e.g., by exercising overdecomposition very large performance gains are possible without modification of the code, see also the figure on the next page. Though such improvements are problem-, and hardware-dependent, there is always an optimum spot that yields a significant improvement. Overdecomposition and writing a parallel code that enables overdecomposition are always advised by the Charm++ developers, as this universally leads to more flexibility and better performance. This is partly due to more efficient load balancing (using smaller work units), and partly due to the smaller work units using local CPU caches more effectively.

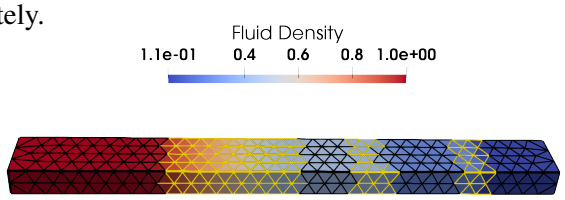
5.2 Solution-adaptive cell-centered discontinuous Galerkin finite element method

A discontinuous Galerkin (DG) finite element method [6] has been implemented with a solution-dependent p-adaptation. The DG method uses the Dubiner basis functions [7, 8] on unstructured tetrahedral grids.



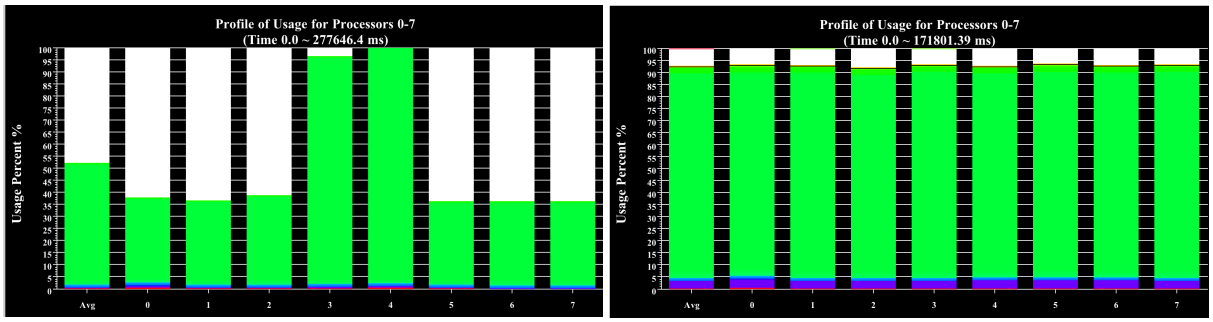
Large performance-improvements due to overdecomposition without load balancing, and most importantly, without any changes to the source code.

These polynomials are orthogonal, meaning that the only non-zero inner products in the L_2 space are their self-projections. This leads to a diagonal mass-matrix, obviating the need for a matrix-inversion and a matrix solver, similar to the lumped-mass node-centered solver, discussed above. Moreover, since the basis for the solutions are orthogonal, they are represented more accurately.

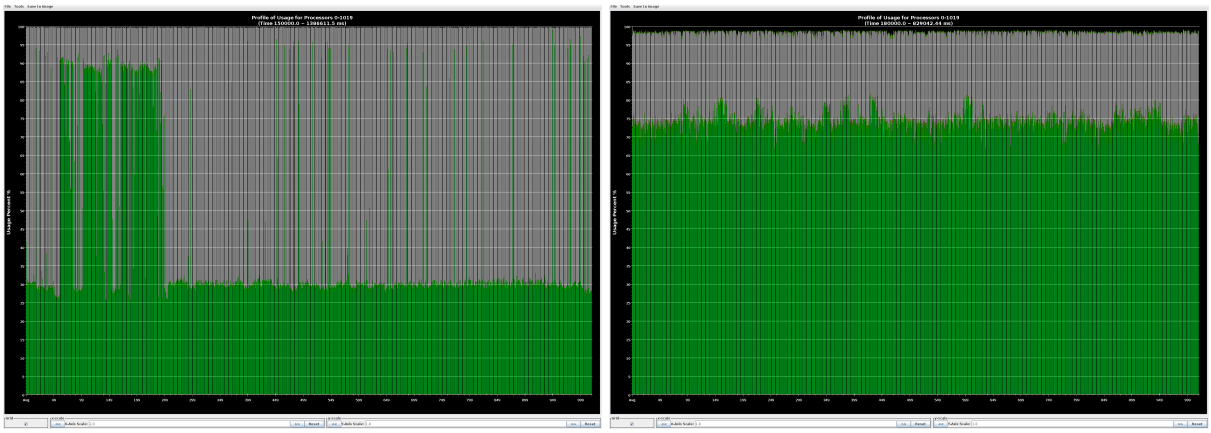


Density contours (top) and line-output (bottom) for the single-material Sod shock tube problem. p-adaptation yields 2nd order DG (yellow) cells near solution gradients and 1st order FV (black) cells otherwise.

The DG method uses a 3rd-order Runge-Kutta time integrator [9] and a total-variation-diminishing Superbee slope limiter to obtain stable solutions in the presence of discontinuities such as shock waves. This is combined with a solution-dependent polynomial, p–adaptation, strategy. The basic idea of p-adaptation involves switching to lower-order (hence cheaper) polynomial basis in areas of near-constant solution states. This can tremendously reduce computation costs without sacrificing accuracy, which stays efficient even in parallel when combined with a load-balancing strategy. The key component of an accurate *and* efficient p-adaptive algorithm is the indicator function [10] used for polynomial refinement or de-refinement.



Processor utilization without (left) and with (right) load balancing employing the p–adaptive DG algorithm on a laptop with 8 CPU cores. Left: traditional MPI-style computation without load balancing, right: load balancing using Charm++'s automatic object migration. The green areas show active times, and white show idle times. Even in this case on a laptop load balancing dramatically improves processor utilization and increases computational speed by almost 2x.

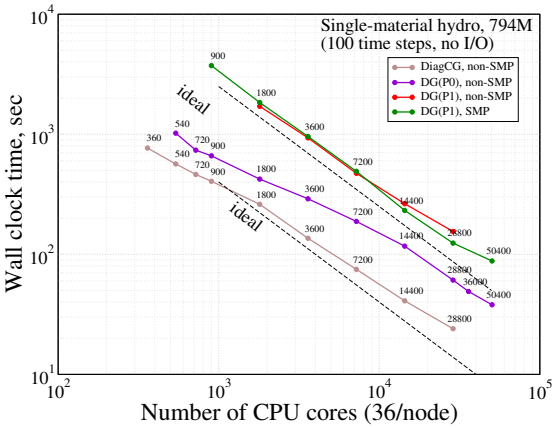


Processor utilization without (left) and with (right) load balancing employing the p–adaptive DG algorithm on a 120-million-cell mesh on 1K CPUs on *grizzly*. The green areas show active times, and white show idle times. Without load balancing 20% of the CPUs are at 90% usage, the remainder are at 30% usage. With load balancing all CPUs are at 75% usage, the rest is communication and load balancing costs. In this case the speed-up compared to the unbalanced case is approximately 1.7x.

One of the error estimators we used for local p–refinement is the spectral decay indicator:

$$\eta_e = \frac{||\rho_{e,p} - \rho_{e,p-1}||_{L_2}}{||\rho_{e,p}||_{L_2}},$$

where, the subscript p indicates the high-order solution, and $p-1$ indicates its projection to a lower-order space. This indicator is calculated for each mesh element e. This element is marked for refinement if $\eta_e > 10^{-4}$ and marked for derefinement if $\eta_e < 10^{-8}$. Significant performance gains are seen if this p–refinement is used with dynamic load balancing. As an example, a speed-up of around 1.6X was observed for our 3–level p-adaptive method, running on 64 compute cores. Strong and weak scaling studies were also performed on *grizzly*. The figure on the right quantifies the excellent scalability of the CG and DG hydro solvers in Quinoa.



Strong scaling of our continuous and discontinuous Galerkin finite element hydro solvers on *grizzly*. *SMP* here means that Charm++ is running in *SMP mode*, which is Charm++'s equivalent of MPI+X where message-based communication is only used on the network, i.e., across compute nodes and not within nodes.

5.3 Reconstructed discontinuous Galerkin finite element method for multi-material flows

We also explored the extension of a reconstructed discontinuous Galerkin (rDG) method to extend the existing DG methods to multi-material shock hydrodynamics. The velocity equilibrium multi-material equations [11] are considered in this work. The Eulerian form of this multi-material system of equations can be expressed as,

$$\frac{\partial U}{\partial t} + \frac{\partial F_j}{\partial x_j} + D = S,$$

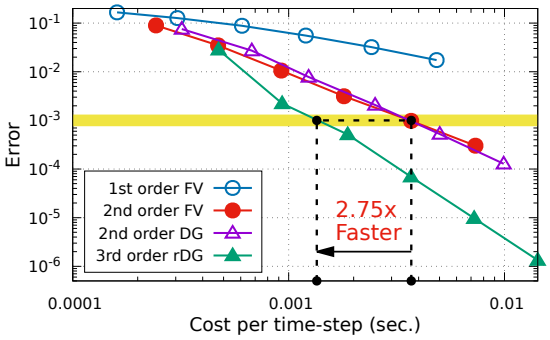
$$U = \begin{bmatrix} \alpha_k \\ \alpha_k \rho_k \\ \overline{\rho} u_i \\ \alpha_k \rho_k E_k \end{bmatrix}, \quad F_j = \begin{bmatrix} 0 \\ \alpha_k \rho_k u_j \\ \overline{\rho} u_i u_j + \overline{p} \delta_{ij} \\ \alpha_k \rho_k E_k u_j \end{bmatrix}, \quad D = \begin{bmatrix} u_j \frac{\partial \alpha_k}{\partial x_j} \\ 0 \\ 0 \\ \alpha_k p_k \frac{\partial u_j}{\partial x_j} + Y_k u_j \frac{\partial \overline{p}}{\partial x_j} \end{bmatrix}, \quad S = \begin{bmatrix} S_{\alpha,k} \\ 0 \\ 0 \\ -\overline{p} S_{\alpha,k} \end{bmatrix},$$

where $k = 1, 2, \ldots, m$ and m is the number of materials. α_k is the volume-fraction of material-k. The overbar, $\bar{\cdot}$, represents bulk material properties such as density $\overline{\rho}$, pressure $\overline{p}$, and internal energy $\overline{\rho e}$. The specific total energy of material k is, $E_k = e_k + u_j u_j/2$. $Y_k = \alpha_k \rho_k / \overline{\rho}$ is the mass fraction of material k. The source term $S_{\alpha,k}$ corresponds to the volume-fraction redistribution due to compaction, owing to the different degrees of compressibility of the different materials. It has the effect of a finite-rate pressure-relaxation. Note that although the material total energy equations are written in a non-conservative form, the mixture total energy equation ($\rho E = \sum_k \alpha_k \rho_k E_k$) is a conservative equation, implying that total energy is conserved. A finite pressure-relaxation is used as a means to redistribute volume changes due to compression (shocks) and expansions (rarefaction waves) between materials depending on their compressibilities. The pressure-relaxation term in the form proposed [12] is used. The volume change redistribution is given as,

$$S_{\alpha,k} = \frac{1}{\tau}(p_k - p^*)\frac{\alpha_k}{\mathcal{K}_k}, \quad \text{where } p^* = \frac{\sum_k \left(p_k \frac{\alpha_k}{\mathcal{K}_k}\right)}{\sum_k \frac{\alpha_k}{\mathcal{K}_k}}, \quad \text{and } \tau = c_\tau \max_k \left(\frac{h}{a_k}\right).$$

where p^* is the pressure that the multi-material cell is expected to equilibriate at, if given sufficient time. $\mathcal{K}_k = \rho_k a_k^2$ is the bulk modulus of material k, and τ is the pressure-equilibration time-scale.

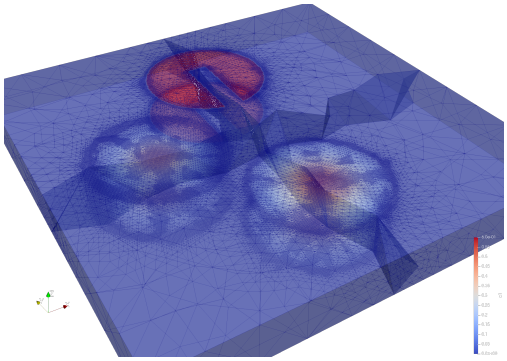
Here the constant c_τ is an adjustable parameter in the pressure relaxation time-scale which affects the pressure equilibration rate of the materials in mixed cells. Setting c_τ to infinity amounts to no pressure relaxation. If c_τ is set to a very small number, stiff pressure relaxation is implied, and an operator-splitting procedure is required for solving this stiff-relaxed system. Here, a value of $c_\tau = \frac{1}{5}\rho_{\min,e}/\rho_{\max,e}$ is used, so that the formulation remains problem-independent. The order of accuracy and the reduction of numerical error with increasing computational cost per time step (polynomial order) was verified to be 3rd-order, indicating significant cost savings for a desired error level. Numerical experiments on problems such as a water-air shock tube and a 3-material shock tube demonstrate consistent and monotonic solutions for shocked problems with materials having widely varying bulk modulii, see figure on the next page. The rDG method has great potential to improve solution accuracy, both, near material interfaces, as well as in single-material regions, for a lower computational costs.



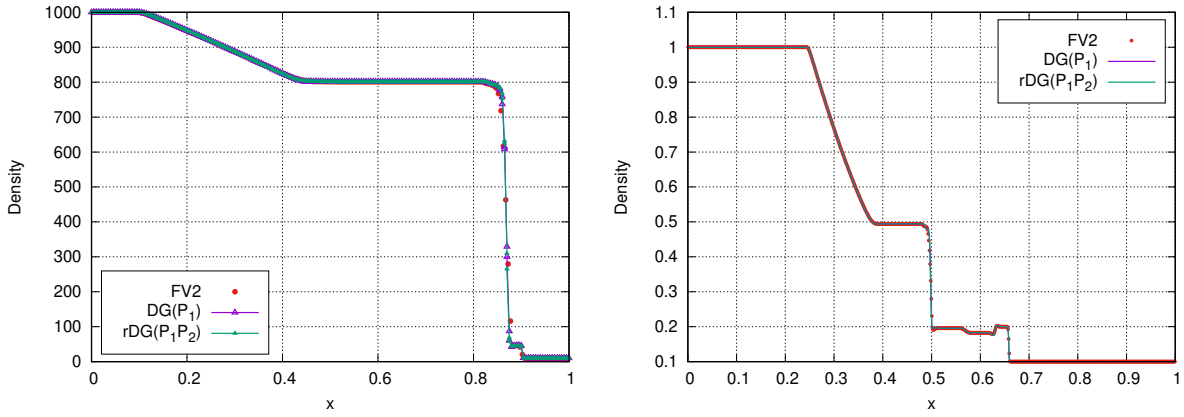
Error-reduction with mesh refinement for multi-material test. Black dashed-line indicates how rDG is 2.75X faster to achieve an accuracy of 10^{-3} (yellow-bar)

6 Adaptive mesh refinement

The adaptive mesh refinement (AMR) algorithm closely follows [13]. This is a mesh refinement and derefinement algorithm specific to tetrahedra-only meshes. Pure-tetrahedra meshes are popular in aerospace engineering because tetrahedra can be used to fill arbitrarily complex geometries and high-quality, parallel, and fully automatic mesh generators have been extensively researched and available. The AMR algorithm in [13] always yields a conforming mesh (without hanging nodes) and never de-refines the mesh beyond the original (coarsest) mesh. The former enables software modularity, since the solver does not need to be specialized due to mesh refinement, and the latter makes enforcing physical conservation properties during mesh refinement straightforward.



Adaptive mesh refinement test problem, after taking 4 refinement steps, on 4 CPUs, correctly detecting large gradients based on the solution gradients and yielding a conforming mesh.



Fluid density for multi-material problems, computed by 3 different numerical methods: FV2: 2nd-order finite volumes, DG(P_1): 2nd-order DG, rDG(P_1P_2): 3rd-order reconstructed DG. (*Left*) liquid-gas shock tube test problem; (*right*) 3-material shock tube.

We first implemented the AMR algorithm in serial to ensure we fully understand the algorithm logic and to enable the maximum degree of independence of the AMR algorithm from its client code, the rest of the solver. This helped us defined a clean, simple, and easy to use interface to the AMR *library*. We then connected the AMR library into the rest of the Quinoa code in a way that the AMR library does not know anything about parallel computing or Charm++, as it only works on a partition of the mesh and is independent of communication primitives or any parallel code. The glue code that uses the AMR library is an asynchronous distributed-memory-parallel algorithm, relying on Charm++'s SDAG constructs, to enable overlap of computational tasks and communication. The final parallel AMR algorithm is also an *iterative* one, since it must ensure that individual decisions for refinement or derefinement, made by independent CPUs on their mesh partitions, connect their partial meshes along their interfaces in a way that yields conforming meshes.

At this time, the AMR algorithm is not yet fully functional: derefinement is incomplete in parallel.

7 Subcontracts

In FY2 we have set up 2 subcontracts within this project with

1. *Charmworks, Inc.*, to get help on improving computational performance and load balancing with the Charm++ runtime system, and
2. *North Carolina State University*, to learn about, initiate research collaboration in, and develop a capability of modern discontinuous Galerkin finite element methods.

Both subcontracts greatly helped the project and we have established two avenues for external collaboration.

7.1 Charmworks subcontract

Our subcontract with Charmworks has set out to include work in the following 4 areas:

1. *Startup performance.* Instrument, analyze, and optimize startup performance, e.g., I/O and parallel communication performance. Identify potential issues and suggest performance improvements.
2. *Solution performance.* Instrument, analyze, and optimize per-time-step performance. Identify potential issues and suggest performance improvements.
3. *Fault tolerance.* Setup tests for migration methods using checkpoints with the same, more, or fewer CPUs, compared to the initial number of CPUs, using one of Charm++'s built-in load-balancers, e.g., RotateLB. Assist in developing solutions to issues identified, and demonstrate fault tolerance against hardware failure during runtime.

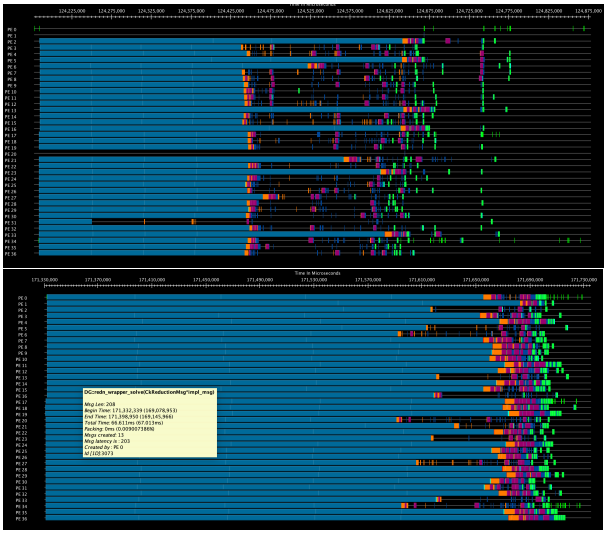
4. *Load balancing with and without adaptive mesh refinement (AMR).* With and without exercising AMR, analyze execution behavior, identify performance deficiencies, and test Charm++'s built-in dynamic remapping strategies. Without sacrificing generality, to the extent as necessary and possible, fine-tune existing strategies to Quinoa, and/or develop a custom application-specific dynamic load-balancing strategy.

Together with Charmworks we have analyzed both the startup and time-stepping performance by profiling their individual steps and tasks, using Charm++'s Projection tool. As a result, we have identified a major performance bottle-neck during problem setup. With the help of Neil Carlson, who suggested a new algorithm, have completely re-written the crucial step. This eliminated 3 all-to-all communication steps and replaced them by a new 2-step algorithm that only involves cheaper and more scalable point-to-point communication steps, working on a smaller data set compared to the previous solution. As a result, the inefficient setup step became practically undetectable by the performance analysis tool. Testing here used a DG method on a 18-million-cell computational mesh on 10 *grizzly* compute nodes, equivalent to 360 CPUs, see also the figure on the next page.

Besides performance analysis and improvement work with the Charmworks employees, we have also received extensive and crucial help regarding the proper use of Charm++'s *bound arrays*, which are distributed Charm++ arrays, holding arbitrary data, e.g., parts of the mesh and the solution, but which they migrate together during load balancing.

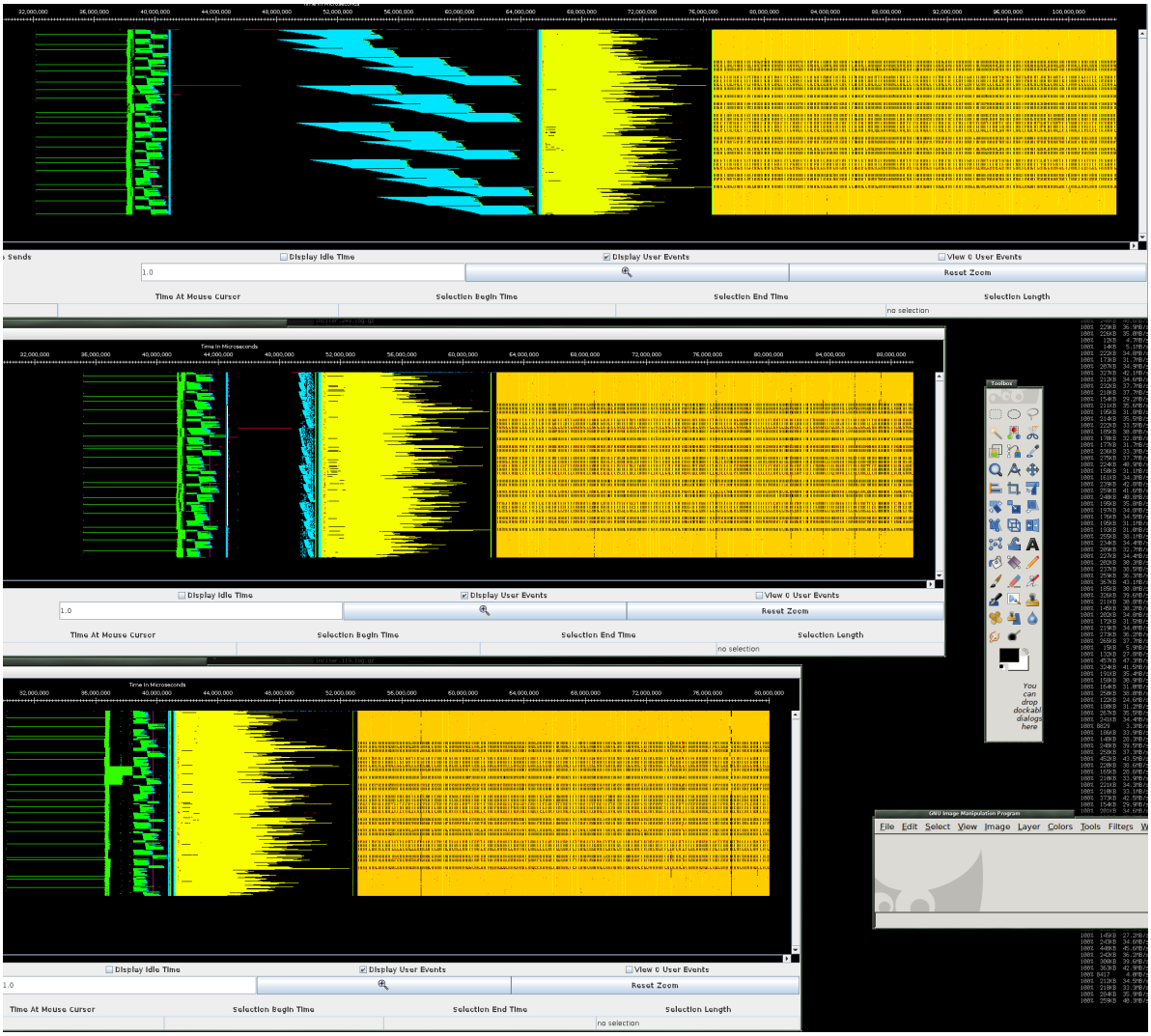
We have also received help implementing the checkpoint/restart functionality in Quinoa. This is now automatically regression-tested in both serial and parallel. In the future, this will also be used for checkpoints and restarts where the number of partitions saved does *not* equal the number of CPUs the restart is performed with. This will yield to a usability feature that allows restarting using different number of CPUs. This feature is also the first step towards on-disk and in-memory fault tolerance, which are built-in Charm++ runtime system features, but in Quinoa still need some work to be fully functional.

With the help of the Charmworks employees, we have also ported the code to Charm++'s *SMP*-mode. This is similar to MPI+X, where X denotes some form of threading (e.g., pthreads, OpenMPI, etc.), allowing efficiently running a parallel code on fat nodes (containing many CPUs) connected by a network. In MPI+X or in Charm++'s SMP mode one configures a single (or a few) MPI rank(s) (or logical node) per physical compute node which then spawn their own threads, instead of running *MPI everywhere*. This may lead to significant performance improvements, especially on accelerators and GPUs. In some cases the performance improvements compared to Charm++'s non-SMP mode were 300%.



CPU utilization of a single compute node on *grizzly* during a single time step, exercising the *p*–adaptive DG algorithm, *without* (top) and *with* (bottom) automatic load balancing. Black means idle while the various colors correspond to different tasks (the light blue is the right hand side computations, taking up most of the computational cost, as expected). The image has been captured from *Projections*, Charm++'s performance analysis tool.

Since the AMR functionality is not yet complete, see the section on AMR above, we did not, so far, exercise AMR with Charm++'s load balancing. However, we have also implemented polynomial-degree refinement in 3D using an asynchronous distributed-memory-parallel algorithm using a *p*–adaptive cell-centered discontinuous Galerkin finite element algorithm and exercised it for single-material verification cases on 3D unstructured meshes with Charm++'s load balancing capabilities, see the figure above. We used Charm++'s automatic load balancing framework with our *p*–adaptive DG algorithm with great success. As



Performance implications and improvements of rewriting an inefficient algorithm during Quinoa's problem setup phase, setting up basic communication data structures, required by all types of (i.e., node-, as well as cell-centered) discretization schemes, after mesh partitioning. The figure depicts 3 results as a result of improving the algorithm in 2 steps. In each sub-figure (top, middle, bottom) the horizontal axis is wall-clock time, while the vertical axis is CPU utilization, different colors denoting different tasks taken during setup and time-stepping (dark yellow on the right). The leftmost parts in green and black roughly correspond to parallel distributed read of the computational mesh (not every CPU reads the mesh). The rightmost part in dark yellow denotes time stepping. The bright yellow preceding the dark yellow are setting up additional communication data structures for cell-centered discretizations, e.g., DG methods. Both the leftmost and rightmost parts (green with black, bright and dark yellow) are the same across all 3 sub-figures. The figure shows, in light blue, the initial algorithm on top, involving 3 all-to-all communication steps with plenty of idle CPU time, denoted by black. The middle image shows the effect of a new point-to-point algorithm that works on smaller data, indicating sizable performance improvement as the lengths of the light blue sections, together with their black idle periods, are reduced. The bottom shows the final result, which includes replacing the entire algorithm with the more efficient one, where the light blue section is practically unmeasurable compared to the rest of the tasks. All performance data were collected using Charm++'s Projections tool with an 18-million-cell mesh on 360 CPUs on *grizzly*.

also expected with AMR, *p*–refinement may lead to order-of-magnitude load imbalances across a large distributed problem. We have successfully employed different Charm++ load balancers with *p*–refinement that automatically homogenized computational load, which significantly improved on unbalanced performance. In some cases the improvement was over 200% and this algorithm only involved the two least expensive DG schemes, P0 and P1. We expect the improvement of the adaptive algorithm, involving P0, P1, and P1, even larger. See also the section on the solution-adaptive cell-centered DG algorithm above.

7.2 NCSU subcontract

With our subcontract to NCSU we have worked on discontinuous Galerkin finite element methods for fluids. This involved implementation of multiple variants of the method, testing, and verification on smooth and discontinuous test problems, and exploration of a *p*-refinement algorithm.

We have successfully implemented 5 different DG algorithms: (1) P0, a basic algorithm that is first-order accurate (equivalent to a cell-centered finite volume method), (2) P1, a second-order accurate method, (3) P2, a third-order accurate method, and two adaptive schemes: (4) a basic *p*–adaptive method that can switch between P0 and P1 based on local error estimation, and (5) a more advanced *p*–adaptive method that switches among P0, P1, and P2. All of these methods are implemented in a way that the code allows to easily configure each based on user input. All of these methods are implemented in a distributed-memory-, and task-parallel fashion for 3D unstructured meshes. For more information on the DG methods, see the sections above on the hydrodynamics methods. This work has predominantly been performed by the Aditya Pandare and Weizhao Li, see also the sections below on student involvement. We have also successfully used *p*–adaptation with Charm++'s automatic load balancing. See also the section above on the Charmworks subcontract.

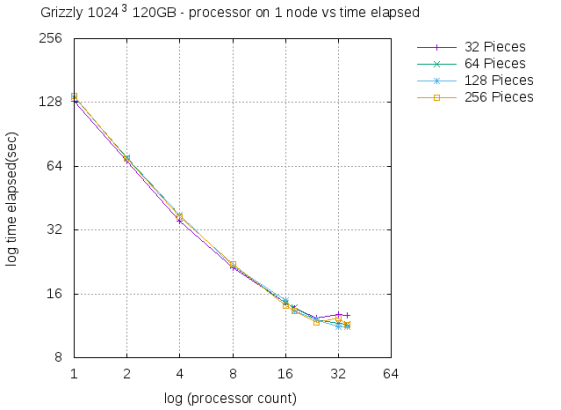
8 Student involvement

During this project we have worked with several students, whose work is described next.

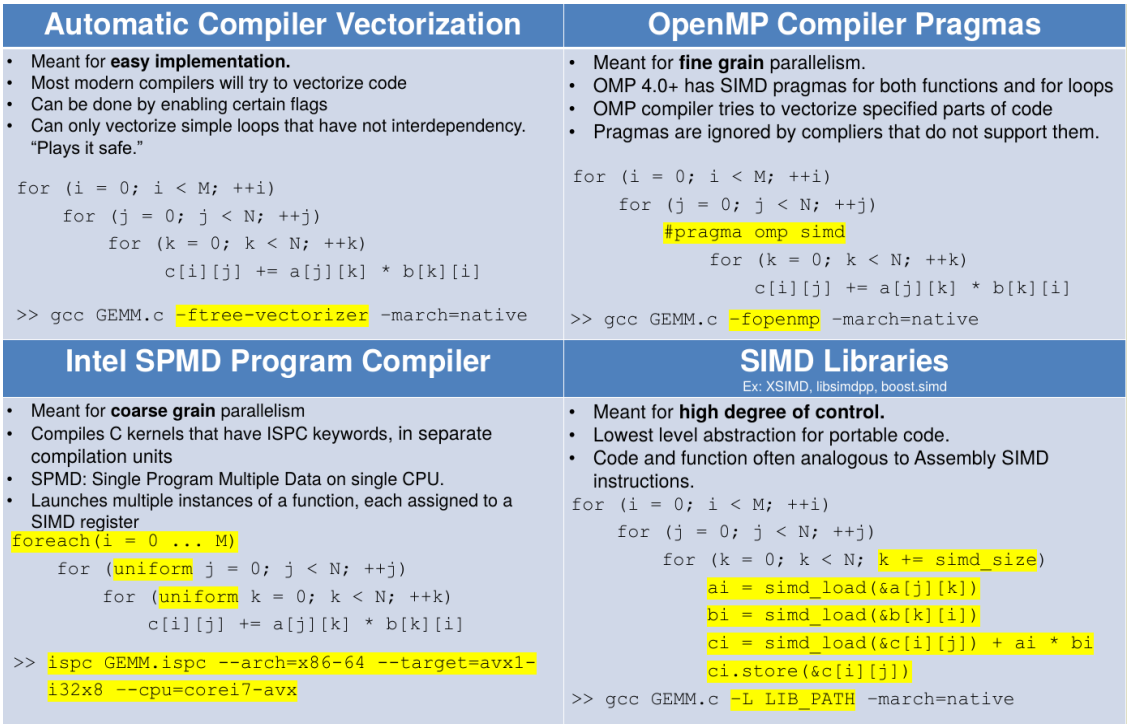
8.1 Aditya Pakki, University of Utah

Aditya was a *Data Science at Scale* summer student and implemented our ROOT writer capability. ROOT is a software package developed by CERN for visualization and data analysis of very large, i.e., petabyte-scale, data sets. The library has been developed for decades and it is the de-facto standard in experimental particle physics. Besides implementing a ROOT file format reader/writer and associated regression tests, Aditya has written code for assessing ROOT's capabilities on our *grizzly* cluster using a large turbulence simulation dataset from Daniel Livesecu and Denis Aslangil.

We usually use ParaView for visualization. However, we are also interested in alternative ways of data analysis. The reason we are interested in using ROOT for data analysis, because it can work with very large data sets that may be distributed across multiple geographical locations and even behind multiple server protocols. Beside using an interactive graphical user interface, data analysis in ROOT may also be driven by a command-line interpreter that understand C++ code in a read-eval-print-loop (REPL) fashion, with access to a large library of various statistical data analysis functionality. In essence, ROOT is interactively programmable, thus encourages easy prototyping the visualization and data analysis code or scripting. Users of Quinoa may need to visualize very large simulation data in the future where the user knows



Data analysis with ROOT in parallel on a 1024^3-cell, turbulence dataset from direct numerical simulation of the homogeneous Rayleigh-Taylor instability by Daniel Livescu and Denis Aslangil. The ROOT code here computed the time-evolution of the turbulent kinetic energy from multiple data files in parallel, without having to visualize the data but having to churn through gigabytes of it.

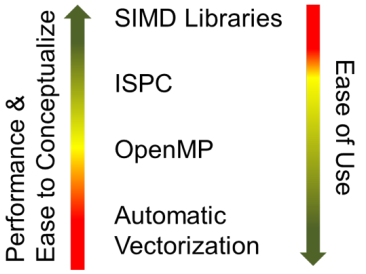


High-level view of our findings and how code looks like with the vectorization abstractions explored.

exactly how to extract some derived quantity from the overall data set but *not* interested in or cannot load the intermediate raw data, but wants to write code to get the result. ROOT allows a potentially very different workflow compared to what we usually do with physics simulations, which may be an interesting alternative for very large simulation data sets in the future.

8.2 Gary Collins, University of Tennessee, Joshua Barnett, Georgia Institute of Technology

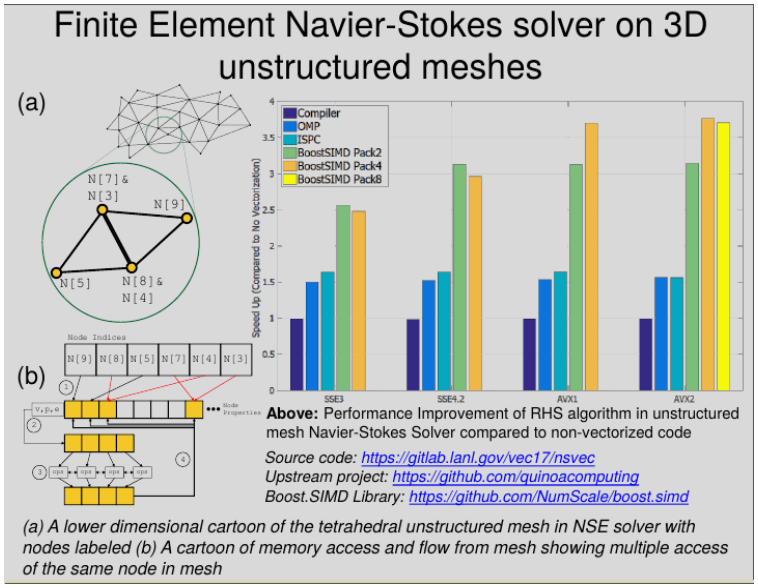
Gary and Joshua were two XCP's *Computational Physics Summer School* students who worked on exploring vectorization techniques. The goal for the summer project was to explore various software abstractions available for vectorizing code and learn what works best for solving the Navier-Stokes equation on unstructured meshes using our node-centered continuous Galerkin finite element algorithm. We set out to explore and answer the following questions. Without vectorization, roughly 90% of hardware performance is left on the table (with Xeon Phi's, e.g., Knights Landing series). Most compilers do vectorize automatically if they can, but do they do it for production-style code and for unstructured-mesh algorithms for non-trivial nested loops full of indirect addressing? Besides the performance increase using vectorization, what are the costs regarding portability, code readability, and maintainability?



The promise of vectorization abstractions.

Besides working on our Navier-Stokes solver, Gary and Joshua have also analyzed the theoretical peak of simpler, linear algebra algorithms, involving matrices and vectors, such as $\mathbf{b} = k \cdot \mathbf{a}$, $\mathbf{C} = \mathbf{A}[\mathbf{b}_1, \mathbf{b}_2, \ldots]$, and have also vectorized a Schrödinger average-atom solver with co-mentor Ondřej Čertík.

Gary and Joshua worked on a simplified code that represented the important features of Quinoa's node-centered continuous Galerkin finite element algorithm using 3D unstructured meshes on a single CPU. They explored vectorization using compiler auto-vectorization, OpenMP 4.0 (no threading only SIMD), Intel's

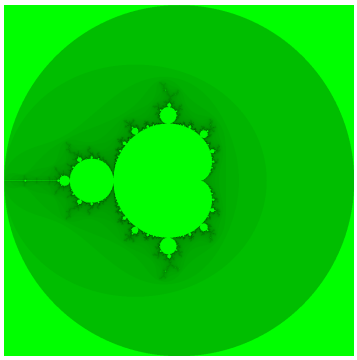


Example vectorization performance gains after vectorizing our node-centered continuous Galerkin finite element unstructured-mesh hydro solver using various abstractions for vectorization.

SPMD Program Compiler (ISPC), as well as 3 C++ libraries: XSIMD, libsimdpp, and Boost.SIMD. A high-level view of our findings, how code looks like with the various abstractions, and example performance gains on our unstructured-mesh finite element solver are illustrated in the figures above. Our overall findings on the use of vectorization abstractions are as follows:

- The compiler will *not* auto-vectorize complex code.
- Easy to conceptualize $\Longleftrightarrow$ easy to use (libraries).
- Libraries and ISPC require new code.
- Automatic vectorization and OpenMP are limited by the compiler.
- Libraries give the best performance but most are intrusive, i.e., require significant code.

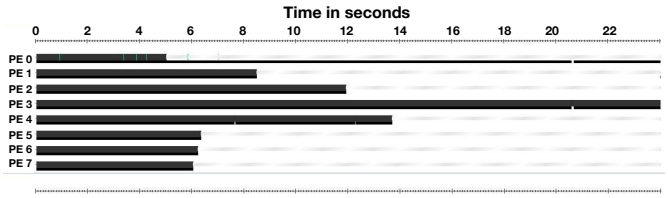
8.3 Aditya Pandare, Weizhao Li, North Carolina State University



Mandelbrot set, as a toy problem, used to learn parallel programming in Charm++.

We have started collaborating with Prof. Hong Luo's research group at North Carolina State University (NCSU) in the 2nd FY. Aditya was a 4th-year PhD student at the time and was researching discontinuous Galerkin finite element methods for compressible hydrodynamics for single-, and multi-material problems. Aditya has started out by learning programming in Charm++ by parallelizing a small Mandelbrot set generator[1] using Charm++. He also exercised Charm++'s load balancers and verified them in action using Charm++'s performance analysis tool, *Projections*. Generating a Mandelbrot set was a toy-project that does not require parallel communication in a complex fashion as a flow solver does, thus was very a useful way of learning the basics of programming in Charm++.

[1] https://github.com/jbakosi/mandelbrot



CPU utilization (black for CPU at work, white for CPU being idle) vs. time for the calculation of a Mandelbrot diagram with 8 CPUs. Top row shows the CPU load with regular parallelization. Bottom row shows CPU load when applying Charm++ ***without any additional code for load balancing***.

By only applying Charm++ the calculation makes better utilization of CPUs and speeds up by a factor of 2.4.

Aditya continued in Quinoa by implementing a lot of low-level infrastructure-code required for cell-centered hydrodynamics schemes. This involve new derived data structures [14] (that were not needed for a node-centered scheme), I/O-related routines, boundary conditions on faces (instead of mesh nodes), as well as setting up parallel communication primitives as required by cell-centered methods and discontinuous Galerkin schemes. As everything else, this involved asynchronous distributed-memory-parallel programming and using Charm++'s SDAG constructs and associated debugging. As a result we now have multiple hydrodynamics schemes in the code, including modern DG schemes. About a year ago, when Aditya finished his PhD, we have hired him as a LANL postdoc and he is currently working on research and development of multi-material DG methods and their implementation into Quinoa.

Weizhao is another one of Prof. Luo's student we have started working with in the last FY of the project. He continues Aditya's work on DG methods by exploring solution adaptation. DG methods can represent a non-linear solution locally within a single computational cell by increasing the order of the underlying approximation polynomial, p–adaptation. By selecting the order differently based on the local error estimated in the numerical solution, one can design a method that spends as much compute resources over a given cell as needed and not more. This requires estimating the local numerical error and devising strategies on how to switch among various schemes. Weizhao started out by implementing a 2nd-, and a 3rd-order accurate DG method for single-material flows in 3D in parallel in Charm++. Then he started exploring p-adaptation by first devising an adaptive method that can switch between 2, then another one that can switch between 3 different DG methods. At the end of FY3 this adaptive scheme now works in parallel and we have also exercised it with Charm++'s automatic load balancing with great success. This p–adaptive method appears to be a very promising method going forward. More details on the DG methods and load balancing with p–refinement are given in previous sections. Our new collaboration established during this project with Prof. Luo's group at NCSU has been extremely fruitful and also feeds into the LANL future pipeline.

9 Publications

The following journal papers, based on this project, have been published or are in progress:

1. On the parallel implementation of our node-centered continuous Galerkin finite element algorithm for compressible hydrodynamics of single-material flows, detailing the asynchronous task-parallel implementation and scalability to large computers, for the Journal of Parallel and Distributed Computing.
2. On a new algorithm, developed for compressible hydrodynamics of single-material flows, using a discontinuous Galerkin finite element method, for the journal Computers & Fluids.
3. On the extension of the discontinuous Galerkin algorithm to a solution-adaptive one and its asynchronous task-parallel implementation, utilizing Charm++'s automatic load-balancing.
4. On a new algorithm for compressible multi-material flows, using a reconstructed discontinuous Galerkin finite element method, under review in International Journal for Numerical Methods in Fluids.
5. On a new algorithm, developed for compressible hydrodynamics of multi-material flows, using a reconstructed discontinuous Galerkin finite element method, its parallel implementation, and its scalability with Charm++'s automatic load balancing, for the Journal of Computational Physics.
6. On the asynchronous, distributed-memory-parallel implementation of the adaptive mesh refinement algorithm for the Journal of Parallel and Distributed Computing.
7. On a new scheme for high-speed compressible flows, published in the journal Shock Waves: A.K.

Pandare and H. Luo, J. Bakosi", *An enhanced AUSM+-up scheme for high-speed compressible two-phase flows on hybrid grids*, Shock Waves, 2018.

10 Conferences

We have presented material at the following meetings and conferences:

1. J. Bakosi, R.F. Bird, C. Junghans, R.S. Pavel, J. Waltz, *Quinoa: Adaptive Computational Fluid Dynamics*, 15th Annual Workshop on Charm++ and its Applications, Urbana-Champaign, IL, 2017.
2. J. Bakosi, R.F. Bird, C. Junghans, A.K. Pandare, H. Luo, *Concept-based runtime polymorphism with Charm++ chare arrays using value semantics*, 16th Annual Workshop on Charm++ and its Applications, Urbana-Champaign, IL, 2018.
3. A.K. Pandare, J. Bakosi, H. Luo, *Progress towards development of discontinuous Galerkin finite-element methods for compressible flows using Charm++*, 16th Annual Workshop on Charm++ and its Applications, Urbana-Champaign, IL, 2018.
4. J. Bakosi, R.F. Bird, C. Junghans, *Quinoa: Adaptive Hydrodynamics on Charm++*, Applied Computer Science and Programming Models/Co-Design Meeting, Albuquerque, NM, 2018.
5. W. Li, A.K. Pandare, J. Bakosi, H. Luo, *Adaptive Discontinuous Galerkin Method for Compressible Flow Using Charm++*, 17th Annual Workshop on Charm++ and its Applications, Urbana-Champaign, IL, 2019.
6. A.K. Pandare, J. Bakosi, J. Waltz, *Discontinuous Galerkin Methods for Compressible Multi-Material Flows*, 20th International Conference on Fluid Flow Problems (FEF-2019), Chicago, IL, 2019.
7. J. Bakosi, R.F. Bird, C. Junghans, A.K. Pandare, J. Waltz, W. Li, H. Luo, E. Bohm, E. Mikida, E. Ramos, L. Kale, *Adaptive Large-Scale Hydrodynamics using Charm++*, Institutional Computing User Group Meeting, LANL, 2019.
8. J. Waltz, A.K. Pandare, J. Bakosi, *A Finite Element ALE Method for Multi-Material Flows*, 20th International Conference on Finite Elements in Fluids, Chicago, Illinois, March 31-April 3, 2019.
9. A.K. Pandare, J. Waltz, J. Bakosi, *A Discontinuous Galerkin method for Non-Equilibrium Multi-Material Flows on Unstructured Grids*, American Institute of Aeronautics and Astronautics (AIAA) Science and Technology Forum and Exposition, 2019.
10. J. Waltz, A.K. Pandare, J. Bakosi, *A direct finite element ALE method for non-equilibrium multi-material flows*, American Institute of Aeronautics and Astronautics (AIAA) Science and Technology Forum and Exposition, 2019.
11. W. Li, H. Luo, A.K. Pandare, J. Bakosi, *A p-adaptive Discontinuous Galerkin Method for Compressible Flows Using Charm++*, American Institute of Aeronautics and Astronautics (AIAA) Science and Technology Forum and Exposition, 2019.

11 Posters

We have prepared and presented the following posters:

1. J. Barnett, G. Collins, J. Bakosi, O. Čertík *Vectorize! Bridging the Performance–Productivity Gap of Vectorization*, X Computational Physics Division Summer School, 2017, LA-UR-17-29526.
2. A.K. Pandare, J. Waltz, J. Bakosi, *Multi-Material Shock Hydrodynamics using a Reconstructed Discontinuous Galerkin Method*, LANL Postdoc Symposium 2019, LA-UR-19-28581.

12 Follow-On projects

This project will continue to be funded by 2 (or potentially 3) sources of funding:

1. *"Engineering, Stockpile Assessments, and Responsiveness (ESAR)"* program. This exploratory work, starting in FY20, is targeted on implementing prototype capabilities to enable physics and algorithms

required to fill a capability gap in LANL's Engineering tool-chain in the ESAR program. The targeted physics capability is large-scale engineering-style (as opposed to physics-style) computational fluid dynamics of aerospace-type problems, eventually targeting large-scale (in terms of problem size as well as using large HPC resources) fluid-structure interaction, involving coupled fluid and solid dynamics. POC: Jacob Waltz.

2. A new LDRD project, *"Adaptive high-order finite element ALE methods for multi-material hydrodynamics"*. This is an LDRD-ER, starting in FY20, that will do R&D towards novel hydrodynamics methods for multi-material problems using discontinuous Galerkin methods. This exploratory work whose end-product, a new hydrodynamics method, will be implemented, verified, and tested at large scales and for large problems in the Quinoa code. PI: Jacob Waltz.
3. Charmworks, Inc. have sought us out to collaborate within a Small Business Innovation Research (SBIR) or Small Business Technology Transfer (STTR) Program of the DOE. This proposal is being written with the goal to increase the technology readiness level of Quinoa and to ultimately increase the capability in LANL's Engineering tool-chain with a new production-quality software tool, suitable for large-scale adaptive computational fluid and coupled solid dynamics simulations over complex 3D engineering problem geometries, using large HPC resources effectively. If this proposal is funded, it will give us an opportunity to further develop our software tool and make it significantly more user-friendly.

13 Acknowledgments

Research presented in this report was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number LDRD-20170127-ER.

References

[1] *Charm++:* Parallel programming framework. http://charmplusplus.org.

[2] R. Löhner, K. Morgan, J. Peraire, and M. Vahdati. Finite element flux-corrected transport (FEM–FCT) for the Euler and Navier–Stokes equations. *Int. J. Numer. Meth. Fl.*, 7(10):1093–1109, 1987.

[3] J. Waltz, T.R. Canfield, N.R. Morgan, L.D. Risinger, and J.G. Wohlbier. Verification of a three-dimensional unstructured finite element method using analytic and manufactured solutions. *Computers & Fluids*, 81:57 – 67, 2013.

[4] J. Waltz, T.R. Canfield, N.R. Morgan, L.D. Risinger, and J.G. Wohlbier. Manufactured solutions for the three-dimensional euler equations with relevance to inertial confinement fusion. *J. Comp. Phys.*, 267:196 – 209, 2014.

[5] J. Bakosi, R.F. Bird, and C. Junghans. Quinoa: Adaptive Hydrodynamics on Charm++. In *Applied Computer Science and Programming Models/Co-Design Meeting*, Albuquerque, NM, 2018.

[6] B Cockburn, S Hou, and C-W Shu. The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. iv. the multidimensional case. *Mathematics of Computation*, 54(190):545–581, 1990.

[7] S J Sherwin and G E Karniadakis. A new triangular and tetrahedral basis for high-order (hp) finite element methods. *International Journal for Numerical Methods in Engineering*, 38(22):3775–3802, 1995.

[8] H Luo and C Pozrikidis. A Lobatto interpolation grid in the tetrahedron. *IMA journal of applied mathematics*, 71(2):298–313, 2006.

[9] S. Gottlieb and C-W. Shu. Total variation diminishing Runge-Kutta schemes. *Mathematics of computation of the American Mathematical Society*, 67(221):73–85, 1998.

[10] F Naddei, M de la Llave Plata, V Couaillier, and F Coquel. A comparison of refinement indicators for p-adaptive simulations of steady and unsteady flows using discontinuous Galerkin methods. *Journal of Computational Physics*, 376:508–533, 2019.

[11] M Pelanti and K-M Shyue. A numerical model for multiphase liquid–vapor–gas flows with interfaces and cavitation. *International Journal of Multiphase Flow*, 113:208–230, 2019.

[12] V A Dobrev, T V Kolev, R N Rieben, and V Z Tomov. Multi-material closure model for high-order finite element Lagrangian hydrodynamics. *International Journal for Numerical Methods in Fluids*, 82(10):689–706, 2016.

[13] J. Waltz. Parallel adaptive refinement for unsteady flow calculations on 3D unstructured grids. *Int. J. Numer. Meth. Fl.*, 46(1):37–57, 2004.

[14] J. Waltz. Derived data structure algorithms for unstructured finite element meshes. *Int. J. Numer. Meth. Eng.*, 54(7):945–963, 2002.